```
/*
UTMconsersion.h - Library to convert in UTM coordenates.
TinyTrackGPS v0.6

Copyright © 2019-2021 Francisco Rafael Reyes Carmona.
All rights reserved.

rafael.reyes.carmona@gmail.com

  This file is part of TinyTrackGPS.

  TinyTrackGPS is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  TinyTrackGPS is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with TinyTrackGPS.  If not, see <https://www.gnu.org/licenses/>.
*/

#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#ifndef UTMconversion_h
#define UTMconversion_h

class GPS_UTM {
  private:
    int _h;
    char _letter;
    long _x;
    long _y;

  public:
    GPS_UTM(){
      _h = 0;
      _letter = 'Z';
      _x = 0L;
      _y = 0L;
    };

    void UTM(double lati, double longi) {
  /*
  * Transformación de las coordenadas geográficas a UTM
  */

  /*// Sobre la geometría del delipsoide WGS84
  double a = 6378137.0;
  double b = 6356752.3142;

    float e1 = sqrt((a*a) - (b*b))/a; ///< Excentricidad.
  double e = sqrt(sq(a) - sq(b))/b; ///< Segunda excentricidad.
```

```
61      double e2 = sq(e); ///< al cuadrado. Usaremos esta directamente.
62      double c = sq(a) / b; ///< Radio Polar de Curvatura.
63    */
64
65    // Se realiza las declaraciones para agilizar el calculo a UTM de X e Y.
66    double e2 = 673949675659e-14; ///< Segunda excentricidad al cuadrado.
67    double e2_2 = 336974837829e-14; // (e2 / 2.0)
68    double c = 639959362580397e-8; ///< Radio Polar de Curvatura.
69    double PI_180 = 1745329251994e-14;// (PI / 180.0)
70    double alf = 505462256744e-14; // (0.75 * e2)
71    double bet = 425820155e-13; // ((5.0 / 3.0) * alf * alf)
72    double gam = 16740579e-14; // ((35.0 / 27.0) * alf * alf * alf)
73
74      /// Sobre la longitud y latitud. Conversión de grados decimales a radianes.
75
76      /*
77      * Cálculo del signo de la longitud:
78      *      - Si la longitud está referida al Oeste del meridiano de Greenwich,
79      *        entonces la longitud es negativa (-).
80      *      - Si la longitud está referida al Este del meridiano de Greenwich,
81      *        entonces la longitud es positiva (+).
82      */
83
84      double latRad = lati * PI_180; ///< Latitud en Radianes.
85      double lonRad = longi * PI_180; ///< Longitud en Radianes.
86
87      /// Sobre el huso.
88      //float huso = ((longi + 180.0) / 6.0) + 1.0;  ///< Nos interesa quedarnos solo
    con la parte entera.
89      //_h = (int)huso;
90    _h = (int)((longi + 180.0) / 6.0) + 1;
91
92    // Handle special case of west coast of Norway
93    if ( lati >= 56.0 && lati < 64.0 && longi >= 3.0 && longi < 12.0 ) {
94      _h = 32;
95    }
96
97    // Special zones for Svalbard
98    if ( lati >= 72.0 && lati < 84.0 ) {
99      if ( longi >= 0.0  && longi <  9.0 ) _h = 31;
100     else if ( longi >= 9.0  && longi < 21.0 ) _h = 33;
101     else if ( longi >= 21.0 && longi < 33.0 ) _h = 35;
102     else if ( longi >= 33.0 && longi < 42.0 ) _h = 37;
103   }
104
105     int landa0 = _h * 6 - 183; ///< Cálculo del meridiano central del huso en
    grados.
106     double Dlanda = lonRad - (landa0 * PI_180);  ///< Desplazamiento del punto a
    calcular con respecto al meridiano central del huso.
107
108   if ((84 >= lati) && (lati >= 72))
109     _letter = 'X';
110   else if ((72 > lati) && (lati >= 64))
111     _letter = 'W';
112   else if ((64 > lati) && (lati >= 56))
113     _letter = 'V';
114   else if ((56 > lati) && (lati >= 48))
115     _letter = 'U';
116   else if ((48 > lati) && (lati >= 40))
117     _letter = 'T';
```

```
  else if ((40 > lati) && (lati >= 32))
    _letter = 'S';
  else if ((32 > lati) && (lati >= 24))
    _letter = 'R';
  else if ((24 > lati) && (lati >= 16))
    _letter = 'Q';
  else if ((16 > lati) && (lati >= 8))
    _letter = 'P';
  else if (( 8 > lati) && (lati >= 0))
    _letter = 'N';
  else if (( 0 > lati) && (lati >= -8))
    _letter = 'M';
  else if ((-8> lati) && (lati >= -16))
    _letter = 'L';
  else if ((-16 > lati) && (lati >= -24))
    _letter = 'K';
  else if ((-24 > lati) && (lati >= -32))
    _letter = 'J';
  else if ((-32 > lati) && (lati >= -40))
    _letter = 'H';
  else if ((-40 > lati) && (lati >= -48))
    _letter = 'G';
  else if ((-48 > lati) && (lati >= -56))
    _letter = 'F';
  else if ((-56 > lati) && (lati >= -64))
    _letter = 'E';
  else if ((-64 > lati) && (lati >= -72))
    _letter = 'D';
  else if ((-72 > lati) && (lati >= -80))
    _letter = 'C';
  else
      _letter = 'Z'; // This is here as an error flag to show that the latitude is
outside the UTM limits


  /*!
    * Ecuaciones de Coticchia-Surace para el paso de Geográficas a UTM (Problema
directo);
    */

    /// Cálculo de Parámetros.
    double coslatRad = cos(latRad);
    double coslatRad2 = sq(coslatRad);

    double A = coslatRad * sin(Dlanda);
    double xi = 0.5 * log((1 + A) / (1 - A));
    double n = atan(tan(latRad) / cos(Dlanda)) - latRad;
    double v = (c / sqrt(1 + e2 * coslatRad2)) * 0.9996;
    double z = e2_2 * sq(xi) * coslatRad2;
    double A1 = sin(2 * latRad);
    double A2 = A1 * coslatRad2;
    double J2 = latRad + (A1 / 2.0);
    double J4 = (3.0 * J2 + A2) / 4.0;
    double J6 = (5.0 * J4 + A2 * coslatRad2) / 3.0;
    double Bfi = 0.9996 * c * (latRad - alf * J2 + bet * J4 - gam * J6);

    /*!
    * Cálculo final de coordenadas UTM
    */
  /*
```

```
    Serial.println (" Las coordenadas GPS que se van a transformar son: ");
    Serial.print (" Latitud: "); Serial.println (lati,6);
    Serial.print (" Longitud: "); Serial.println (longi,6);

    Serial.println (" Coordenadas UTM actuales: ");
  Serial.print("UTM: "); Serial.print(_h); Serial.print("
");Serial.println(_letter);
  */
  _x = round(xi * v * (1 + (z / 3.0)) + 500000);
  /*!< 500.000 es el retranqueo que se realiza en cada huso sobre el origen de
    coordenadas en el eje X con el objeto de que no existan coordenadas negativas.
*/

    _y = round(n * v * (1 + z) + Bfi);
  if (lati < 0.0) _y += 10000000;
  /*!< En el caso de latitudes al sur del ecuador, se sumará al valor de Y
10.000.000
    para evitar coordenadas negativas. */

  /*
    Serial.print (" X = "); Serial.print (_x); Serial.println (" (m)");
    Serial.print (" Y = "); Serial.print (_y); Serial.println (" (m)");
  */
    };

  int zone(){
    return _h;
  };

  char band(){
    return _letter;
  };

  long X(){
    return _x;
  };

  long Y(){
    return _y;
  };
};

#endif

```